Computer Science Department

TECHNICAL REPORT

"An Iterative Version of Hopcroft and Tarjan's Planarity Testing Algorithm"

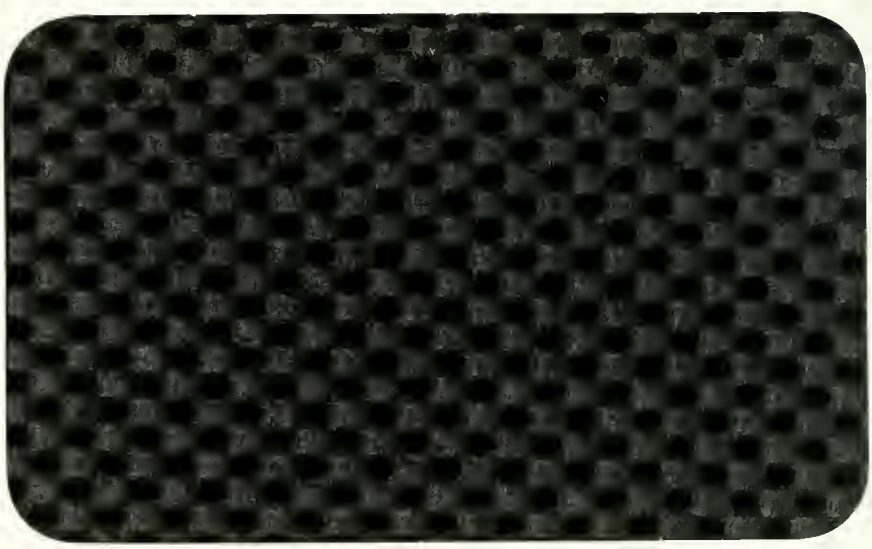
J. Cai

Technical Report # 324

October 1987

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



"An Iterative Version of Hopcroft and Tarjan's Planarity Testing Algorithm"

J. Cait

Technical Report # 324

October 1987

† This work was supported by the Office of Naval Research under contracts N00014-84-K-0444, N00014-85-K-0046 and by Thomson-CSF.

An Iterative Version of Hopcroft and Tarjan's Planarity Testing Algorithm

Jiazhen Cai[1]

Courant Institute, NYU
New York, NY 10012

ABSTRACT

We present a simplified, totally iterative version of Hopcroft and Tarjan's Planarity Testing Algorithm. Interestingly, this algorithm can be derived from a formal problem specification.

• Introduction

In [4], Hopcroft and Tarjan presented the first linear time algorithm for testing planarity of graphs. The main part of their algorithm is to check the planarity of a biconnected component. It finds a cycle in the component and deletes it, leaving a set of disconnected pieces. Then the algorithm recursively checks the planarity of each piece plus the original cycle and determines whether these pieces can be combined to give a whole planar graph. Therefore, this algorithm processes one *cycle* at a time. We find that their discussion and algorithm can be simplified if we consider one *edge* at a time. Our approach allows us to develop a totally iterative version of their algorithm, which is already an iterative version of a method originally proposed by Auslander and Parter [1].

• Problem

Given an undirected graph $G = (V, E)$, we can draw a picture G' on a plane as follows: for each vertex $v \in V$, we draw a distinct node v'; for each edge $(u, v) \in E$, we draw an arc connecting the two nodes u' and v'. We call this arc an *embedding* of the edge (u, v), and call G' an *embedding* of G. For brevity, we will identify edges and paths with their embeddings. If arcs of G' do not cross each other, we say that G' is *planar*. If G has a planar embedding, then we say that G is *planar*. The problem is: how to determine the planarity of a graph G?

1 This work was supported by the Office of Naval Research under contracts N00014-84-K-0444, N00014-85-K-0046 and by Thomson-CSF.

A more precise and complete description of the planarity testing problem can be found in [4, 5].

- **Some observations**

The following facts are important to our discussion:

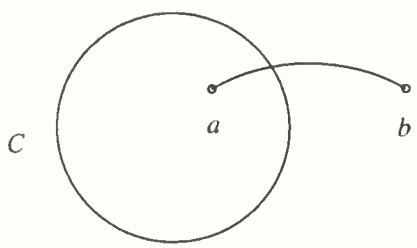


Fig.1

Observation 1. Let C be a simple cycle on the plane; let a be a point inside C and b be a point outside C. Then any curve that joins a and b will cross C. (See Fig. 1)

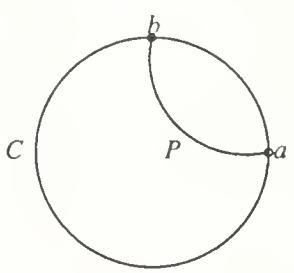


Fig.2

Observation 2. Let G_1 be the undirected graph represented by Fig. 2. Then in any planar embedding of G_1, all the edges of path P must be in the same side of the cycle C.

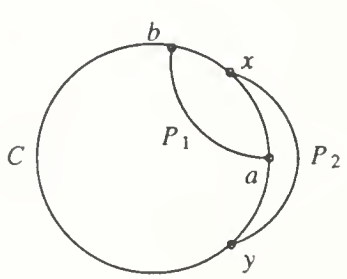


Fig. 3

Observation 3. Let G_2 be the undirected graph represented by Fig. 3. Then in any planar embedding of G_2, the two paths P_1 and P_2 must be in different sides of the cycle C.

All the above observations are intuitively obvious, and can be proved by the Jordan Curve Theorem. (See [3, 8] .)

• **Definitions**

A depth first search [4] will convert G into a directed graph $G^* = (V^*, T, B)$, where V^* is the set of DFS numbers of vertices in V, T is the set of tree edges, B is the set of back edges. In the depth first search, each edge of G is converted into either a tree edge or a back edge. All the tree edges form a DFS forest, and each back edge goes from a vertex of the DFS forest to its ancestor. We will make no distinction between G and G^*, and between V and V^*. We assume that G has no self-loops.

Let $e = [a, b] \in B \cup T$. If $e \in T$, then we use $bridge(e)$ to denote the subgraph that consists of e, the subtree of T rooted at b, and all the back edges that emanate from vertices of this subtree; if $e \in B$, then $bridge(e)$ is defined to be e itself.

Lemma 1: Let e_1 and e_2 be two edges leaving b, and $e_1 \neq e_2$. Then all the common vertices of $bridge(e_1)$ and $bridge(e_2)$ are ancestors of b.

Proof Follows from the properties of depth first search.

We use $attachments(e)$ to denote the set of back edges in $bridge(e)$ that go into proper ancestors of a. Each back edge in $attachments(e)$ is called an *attachment* of e.

We define $low(e)$ to be the lowest ancestor of b in $bridge(e)$, and $low2(e)$ to be the second lowest ancestor of b in $bridge(e)$. More precisely,

$$low(e) = \begin{cases} b & \text{if } e \in B \\ \min / (\{ b \} \cup \{ low([x, y]): [x, y] \in T \cup B, x = b \}) & \text{if } e \in T \end{cases}$$

and

$$low2(e) =$$

$$\begin{cases} b & \text{if } e \in B \\ \min /((\cup/\{ \{low([x,y]), low2([x,y])\}: [x,y] \in T \cup B, x = b \} - \{low(e)\}) \cup \{b\}) & \text{if } e \in T \end{cases}$$

We define the function C on $T \cup B$ as follows:

$$C([v,w]) = \begin{cases} 2w & \text{if } [v,w] \in B \\ 2\, low([v,w]) & \text{if } [v,w] \text{ in } T, \text{ and } low2([v,w]) \geq v \\ 2\, low([v,w]) + 1 & \text{if } [v,w] \text{ in } T, \text{ and } low2([v,w]) < v \end{cases}$$

Then for any two edges e_i and e_j leaving b, we have

1. if $low(e_i) < low(e_j)$, then $C(e_i) < C(e_j)$;

2. if $low(e_i) = low(e_j)$, $\#\{y: [x, y] \in attachments(e_i)\} = 1$, and $\#\{y: [x, y] \in attachments(e_j)\} > 1$, then $C(e_i) < C(e_j)$.

Let $L(e) = [e_1, e_2, ..., e_k]$ be the list of edges leaving b such that for all $i, j = 1..k$, $i < j$ implies $C(e_i) \leq C(e_j)$. We define $cycle(e)$ as follows: if e is a back edge, then $cycle(e) = e$ + the tree path from b to a, otherwise $cycle(e) = cycle(e_1)$. We use $sub(e)$ to denote the subgraph $bridge(e) + cycle(e)$. The edge e is said to be *planar* if $sub(e)$ is planar. If there exists a planar embedding of $sub(e)$ such that all the attachments of e are in the same side of $cycle(e)$, then we say that e is *strongly planar*.

See Fig. 10 for an example of some of these definitions.

Theorem 2: G is planar if and only if each edge in $B \cup T$ is planar.

Proof Obvious.

• Tarjan's approach and our approach

Tarjan's approach considers the following basic problem: Given $cycle(e) = [e, t_1, ..., t_i, b, a_1, ..., a_j]$, where b is a back edge, and given that all the edges leaving the heads of $e, t_1, ..., t_i$ but not contained in $cycle(e)$ are planar, how can we determine the planarity of e?

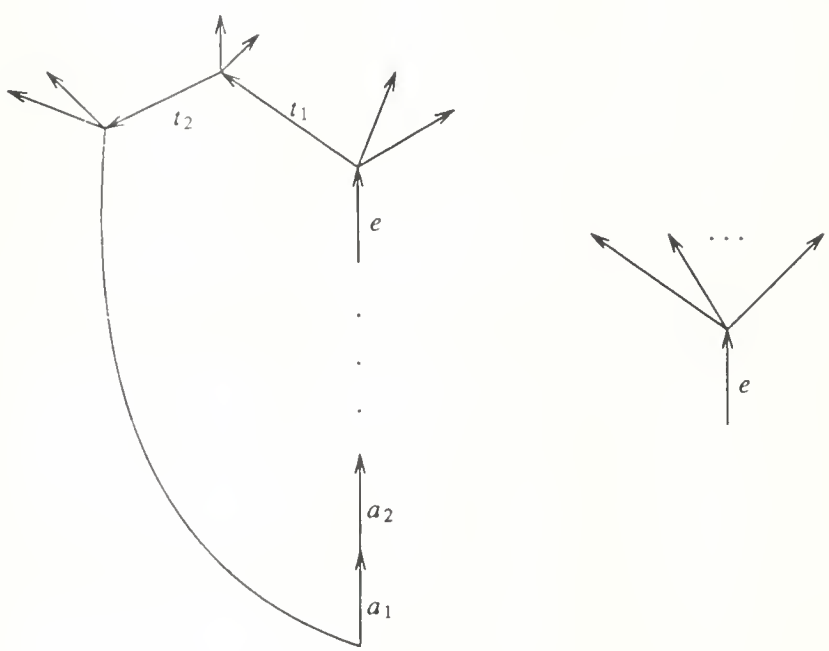


Fig. 4

We find that the following problem is more basic: Suppose that all the edges leaving b are planar, how can we determine the planarity of e? This is a subproblem considered in Hopcroft and Tarjan's approach. All the theorems and techniques which are necessary to solve this problem are already there in [4, 5] , and are reorganized in the following sections.

• **Structure of attachments**

Suppose e is planar. We can divide *attachments* (e) into *block*s. Two attachments of e are in the same block if and only if they are always in the same side of *cycle* (e) in the planar embeddings of *sub* (e).

Let B_1 and B_2 be two blocks of *attachments* (e), $S_1 \subseteq B_1$, $S_2 \subseteq B_2$. S_1 and S_2 are said to be *interlacing* if they are never in the same side of *cycle* (e) in the planar embeddings of *sub* (e).

The special attachment that is on *cycle* (e) forms a block by itself, and this block is not interlacing with other blocks.

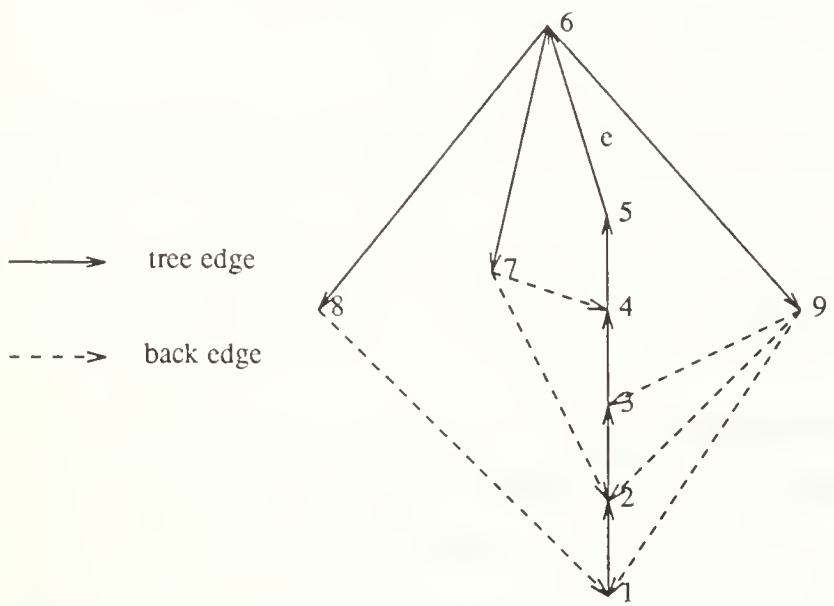


Fig. 5

In Fig 5, *sub* (e) is the whole graph; *bridge* (e) contains all the edges in *sub* (e) except [1, 2], [2, 3], [3, 4], [4, 5]; *cycle* (e) = { [5, 6], [6, 8], [8, 1], [1, 2], [2, 3], [3, 4], [4, 5] }; *attachments* (e) = {[8, 1], [9, 1], [9, 2], [9, 3], [7, 2], [7, 4]}. The edge e is planar, but not strongly planar. *Attachments* (e) can be divided into three blocks: $B_1 = \{[8, 1]\}$, $B_2 = \{[9, 1], [9, 2], [9, 3]\}$, and $B_3 = \{[7, 2], [7, 4]\}$. B_2 and B_3 are interlacing.

Lemma 3: If e is planar, then each block of *attachments*(e) can be interlacing with at most one another block.

Since only the heads of the attachments are involved in our computation, we will represent a block of attachments $A = \{[b_1, a_1], [b_2, a_2], ..., [b_k, a_k]\}$ by a list $B = [a_1, a_2, ..., a_k]$, where we assume that $a_1 \le a_2 \le ... \le a_k$. We say that B is the *list representation* of A. Frequently, we will identify blocks with their list representations. For any list of integers $B = [a_1, a_2, ..., a_k]$, we define $first(B) = a_1$, and $last(B) = a_k$.

For each pair of interlacing blocks B_1 and B_2 of *attachments*(e). We form a pair $[B_1, B_2]$, assuming $[last(B_1), first(B_1)] \le [last(B_2), first(B_2)]$, where $\le$ is the lexicographical ordering. If a block B is not interlacing with any block, we form a pair $[\,[\,], B\,]$.

We say $[A_1, A_2] < [B_1, B_2]$ if $[A_1, A_2] \ne [B_1, B_2]$ and $last(A_2) \le min(first(B_1), first(B_2))$.

Lemma 4: Let $[A_1, A_2]$, $[B_1, B_2]$ be two pairs of blocks of *attachments*(e) such that $[A_1, A_2] \ne [B_1, B_2]$. Then either $[A_1, A_2] < [B_1, B_2]$ or $[B_1, B_2] < [A_1, A_2]$.

Proof See Lemma 5, a) of [5].

Let $S = \{p_1, p_2, ..., p_m\}$ be the set of pairs of blocks of *attachments*(e). By Lemma 4, there is a linear ordering on S, say, $p_1 \le p_2 \le \cdots \le p_m$. We then represent *attachments*(e) by $att(e) = [p_1, ..., p_m]$.

In Fig. 5, $att(e) = [p_1, p_2]$, where $p_1 = [\,[\,], [1]\,]$, $p_2 = [\,[1, 2, 3], [2, 4]\,]$.

Now we are ready to compute att(e).

- **Compute *att*(e) for back edges and dead edges**

If $e = [a, b]$ is a back edge, then its only attachment is itself. Therefore

$$att(e) = [\,[\,], [b]\,]$$

If $e = [a, b]$ is a tree edge such that no edge leave from b, then we say e is *dead*. Dead edges has no attachment:

$$att(e) = [\,]$$

- **Compute *att*(e) from *att*(e_i)**

Now we discuss how to compute $att(e)$ for a tree edge e that is not dead, given that $att(e_i)$ has been computed for each e_i in $L(e)$.

Step 1. We first consider what we should do with the attachments of e_i, where $i = 2..k$. According to Observation 2, all the attachments of e_i must be in the same side of $cycle(e_1)$ in any planar embedding of sub(e). This requires that the blocks of $att(e_i)$ do not interlace with each other. If this is the case, we merge all the blocks of $att(e_i)$ into one intermediate block B_i. See Fig. 6.

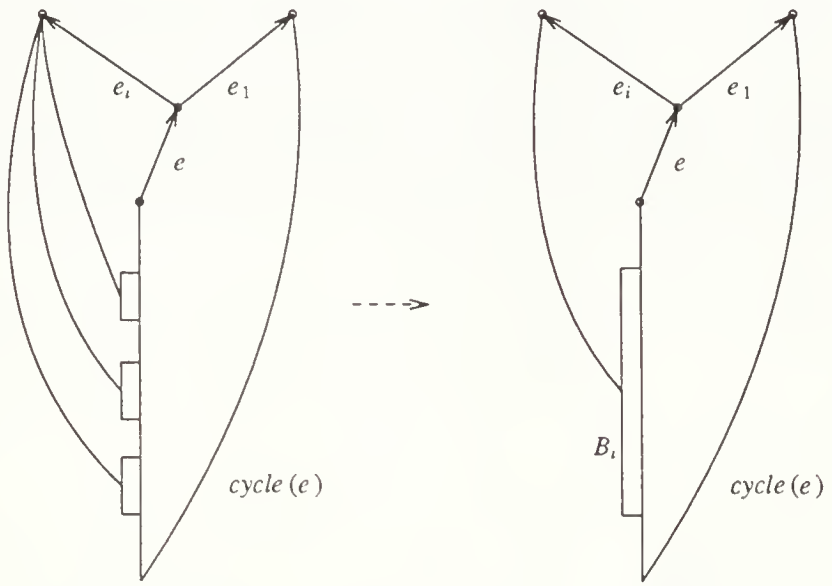


Fig. 6

If we define

$$[u,...,v] + [w,...,x] = \begin{cases} fail & v > w \\ [u,...,v,w,...,x] & otherwise \end{cases}$$

and

$$[X,Y]\ [+]\ [U,V] = \begin{cases} fail & X \neq [\,]\ or\ U \neq [\,] \\ Y+V & otherwise \end{cases}$$

where $[u,...,v]$ and $[w,...,x]$, X, Y, U, V are lists of integers, then

$$B_i = [+] / att(e_i)$$

Step 2. Next we consider what we should do with the attachments of e_1. Let $att(e_1) = P_0 + P_1$, where

$$P_0 = [\ [A,B] \in att(e_1) \mid last(B) \leq low(e_2)\ ]$$

and

$$P_1 = [\ [A,B] \in att(e_1) \mid last(B) > low(e_2)\ ]$$

Also let

$$P_A = [\, A : [A, B] \in P_1 \,]$$

and

$$P_B = [\, B : [A, B] \in P_1 \,]$$

According to Observation 3, all the attachments of e_1 whose heads are higher than $low(e_2)$ must be in the same side of $cycle(e)$ in any planar embedding of $sub(e)$. Therefore all the blocks in P_B must be merged into one intermediate block B_1. As a result, all the blocks in P_A must also be merged into a intermediate block B_0, which will be interlacing with B_1. Of course, B_0 should not contain any element that is higher than $low(e_2)$ in order for e to be planar. This condition will be checked later when we try to merge B_0 with B_2. See Fig 7.

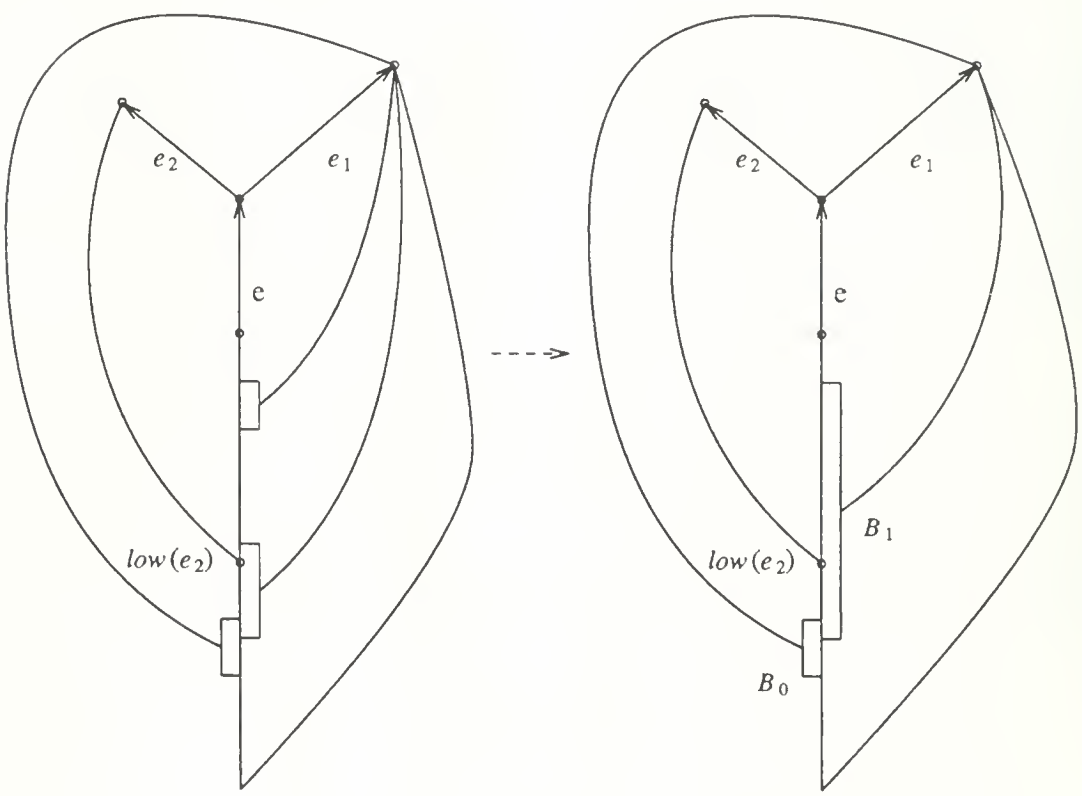


Fig. 7

If we define

$$[X, Y] (+) [U, V] = [X+U, Y+V]$$

then

$$[B_0, B_1] = (+) / P_1$$

Step 3. Now we consider how to combine all the intermediate blocks B_0, ..., B_k with P_0 to obtain *att*(e). First we add the pair $[B_0, B_1]$ to the end of P_0. Then we add $B_2, B_3, ..., B_k$ to P_0 one by one in that order. When B_i, i = 2 .. k, is considered, we check the last pair, say $[A, B]$, of P_0. If *first*$(B_i) \geq$ *last*(B), then B_i is not interlacing with B. We form a new pair $[\ [\ ], B_i\ ]$ and add it to the end of P_0. On the other hand, if *first*$(B_i) <$ *last*(B), then B_i is interlacing with B. In this case, if *first*$(B_i) <$ *last*(A), then B_i and A are also interlacing, and *sub*(e) is not planar; otherwise, we have *first*$(B_i) \geq$ *last*(A) and B_i does not interlace with A, therefore we can merge them into a new block B_i', which is also interlacing with A. If *last*$(B_i') \leq$ *last*(B), then $[B_i', B]$ becomes the last pair of P_0. Otherwise, we have to change the position of B_i' and B.

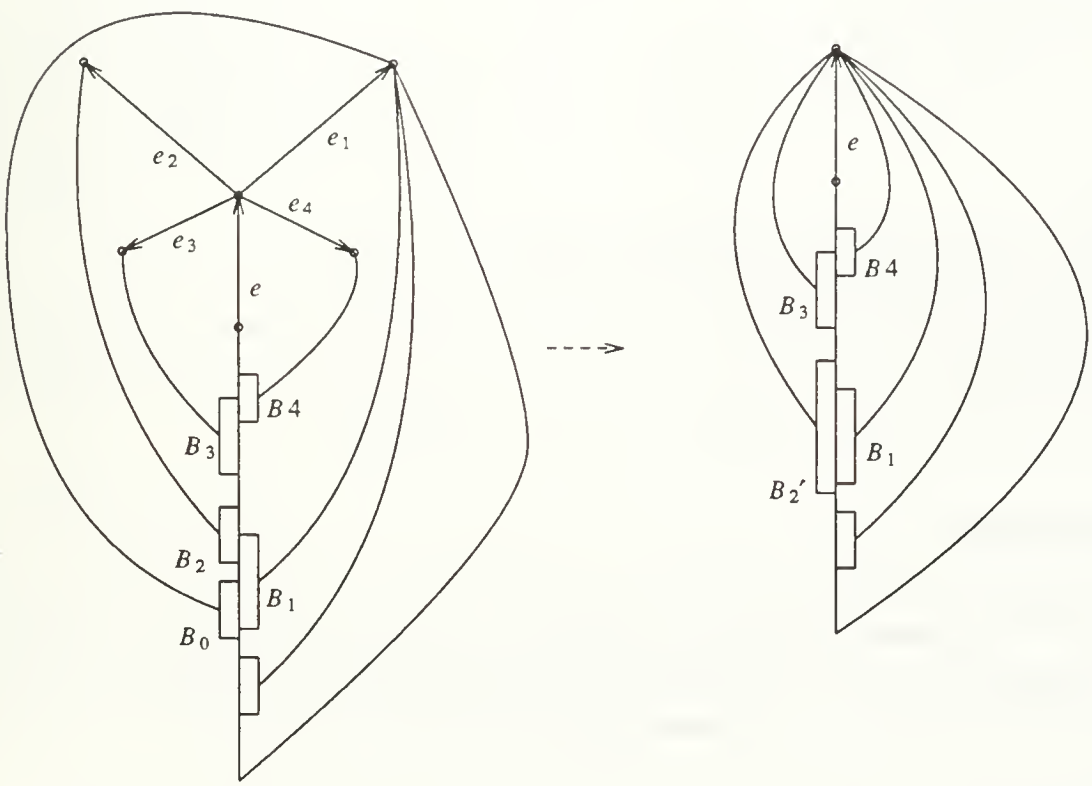


Fig. 8

Fig. 8 shows how the embeddings of $sub(e_i)$, $i = 2, 3, 4$, are added to the embedding of $sub(e_1)$. It also shows that, if B_i does not interlace with the last pair of P_0, then $sub(e_i)$ can be embedded either inside $cycle(e)$ or outside $cycle(e)$.

After all the B_i's are added to P_0, we have to delete b from each block in P_0, since $att(e)$ should contain only proper ancestors of b. We call the resulting list of pairs P_e.

To express the result of this step, we define:

$$pair([[u,...,v],[w,...,x]]) = \begin{cases} [[u,...,v],[w,...,x]] & [v,u] < [x,w] \\ [[w,...,x],[u,...,v]] & otherwise \end{cases}$$

$$[X,Y] <+> B = pair([X+B,Y])$$

$$drop(P) = [\,[A,B] \in P \mid [A,B] \neq [\,[\,],[\,]\,]\,]$$

$$[\,[A_1,B_1],...,[A_i,B_i]\,]\ \{+\}\ B = \begin{cases} [\,[A_1,B_1],...,[A_i,B_i],[\,[\,],B\,]\,] & first(B) \geq last(B_i) \\ [\,[A_1,B_1],...,[A_i,B_i] <+> B\,] & otherwise \end{cases}$$

$$[\,[A,B],...,[C,D]\,] - x = drop([\,pair([A',B']),...,pair([C',D'])\,],$$

where $A', B', ..., D'$ are obtained from $A, B, ..., D$, respectively, by deleting all the elements that are equal to x.

With these notations, we have

$$P_e = (P_0 + \{+\} / [[B_0,B_1],B_2,...,B_k]) - a$$

Theorem 5

i. If the three step computation does not fail, then e is planar;

ii. If $e \in T$ is planar, and e is not dead, then

$$P_e = att(e)$$

iii. If the blocks of P_e do not interlace with each other, then e is strongly planar.

Proof See Appendix.

Thus, the three step computation described above defines a procedure which, given $att(e_i)$ for all $e_i \in L(e)$, returns the value $att(e)$ if e is planar, and returns $fail$ otherwise. We call this procedure $merge(e)$. For a dead edge d, we define $merge(d) = [\,]$.

• .Program

If G is planar, then att is a total function defined on $B \cup T$. According to the above discussion, att must be the minimum solution to the following equation:

$$att = \{ [x, y], [\,[\,], [y]\,]\,]: [x, y] \in B \} \cup \{ [e, merge(e)]: e \in T \mid (\forall t \in L(e) \mid t \in domain\ att) \text{ and } (att(e) \neq fail) \}$$

which has the form $x = f(x)$, where f is a monotone function. Using the techniques discussed in [2, 6, 7] , we can derive the following program from it:

```
(10)  att := {};
      nump := {[e, #L(e)]: e ∈ T};
      new := {[[x, y],[[ ], [y]]]: [x, y] ∈ B} ∪ {[e, [ ]]: e ∈ T | L(e) = [ ]};
      (while ∃ [e, z] ∈ new)
            u := L⁻¹(e);
            if nump(u) = 1 then
                  t := merge(u);
                  if t ≠ fail then
                        new with:= [u, t];
                  end;
            end;
            nump(u) -:= 1;
            new less:= [e, z];
            att with:= [e, z];
      end;
```

in which the att value is first computed for back edges and dead edges, and then computed for other tree edges in a topological order on the DFS tree of G. Let att_0 be the final value of att. According to Theorem 2, G is planar if and only if for all $e \in B \cup T$, we have $e \in domain\ att_0$.

• **Complexity**

In the computation of $merge(e)$, the values of $att(e_i)$, where $e_i \in L(e)$, can be discard after they are used in computing $att(e)$. So we can use the same list operations suggested in [4] to implement our procedure $merge(e)$. I.e., we implement a list of integers as a link list, and implement a pair of lists of integers as a pair of pointers. In this way, each of the operations +, [+], (+), <+>, {+} and *pair* can be implemented in $O(1)$ of time.

Let op be any operation. We use cost(op) to represent the total cost of the operation op in program (10). Since each execution of the operations [+] and (+) reduces the total number of intermediate blocks by one, and initially there are $\#B$ intermediate blocks, where $\#B$ is the

number of back edges, we have $cost([+]) + cost((+)) = O(\#B)$, and the total cost of step 1 and step 2 of *merge* is $O(\#B) + O(\#T) = O(\#E)$.

If $A, B, ..., C, D$ are lists of integers in increasing order, $[A, B] \leq \cdots \leq [C, D]$, and $last(D) \leq x$, then $L - x$ can be implemented in $O(n(L, x))$ time, where $L = [\,[A, B], ..., [C, D]\,]$ and $n(L, x)$ is the number of occurrences of x in L. Let $|L|$ be the total length of blocks in L, then the cost for computing $L - x$ is $O(1) + O(|L| - |L - x|)$. Initially the total length of all the blocks is $\#B$, therefore $cost(-) = O(\#B) + O(\#T) = O(\#E)$. Because the operation $\{+\}$ is executed $O(\#L(e))$ times for each tree edge e, we have $cost(\{+\}) = O(\#E)$. Thus, the total cost of step 3 is also $O(\#E)$.

Therefore the total cost of program (10) = (total cost of *merge*) + (other cost) = $O(\#E) + O(\#E) = O(\#E)$.

• **Summary**

We presented a totally iterative version of Hopcroft and Tarjan's planarity testing algorithm (H-T algorithm). The control structure of our algorithm is so simple that it can be derived from one line formal specification. In H-T algorithm, the edges are visited in an order totally determined by the depth first search. In our algorithm, the edges can be visited in any topological order of the DFS tree. In fact, our algorithm can be viewed as a topological sorting on the DFS tree combined with some information propagation. Also, our algorithm allows some degree of parallelism: all the edges on the frontier of the DFS tree can be processed parallelly. H-T algorithm works on biconnected components of G, we do not need this condition.

The same idea of *dependency graph* mentioned in [4] can be used in our algorithm to actually construct a planar embedding of G, also in linear time.

• **Appendix** Proof of Theorem 5

We prove Theorem 5 by induction.

For backedges and dead edges, we need only prove iii, which is trivially true. Now we consider an edge $e \in T$ with $L(e) = [e_1, ..., e_k]$, where $k > 0$, and assume that Theorem 5 is true for all $e_i \in L(e)$. Also assume the computation of merge does not fail.

According to Step 1, the blocks of $att(e_i)$, $i \in \{2, ..., k\}$ do not interlace with each other. Therefore each $e_i \in L(e)$ is strongly planar, and there is a planar embedding of $sub(e_i)$ such that all the attachments of e_i are inside $cycle(e_i)$. In step 3, we showed how these embeddings can be added to the planar embedding of $sub(e_1)$ to get the planar embedding of $sub(e)$. Therefore e is planar, and i is proved.

In step 3, we also mentioned that if B_i does not interlace with the blocks of P_0, then $sub(e_i)$ can be embedded in either side of $cycle(e)$. Especially, if all the blocks of P_e do not interlace with each other, they can always be embedded inside $cycle(e)$. Thus we proved iii.

In the computation, two small intermediate blocks are merged into a large one only when they are always in the same side of $cycle(e)$ in the planar embeddings of $sub(e)$. Therefore the attachments contained in the same block of P_e are also contained in the same block of $att(e)$. Also in step 3 we showed that for any two different blocks B_1 and B_2 of P_e, there exists an planar embedding of $sub(e)$ such that B_1 and B_2 are in the different sides of $cycle(e)$. This means that attachments of e contained in different blocks of P_e are also contained in different blocks of $att(e)$. Therefore P_e and $att(e)$ have the same blocks. In addition, two blocks form a pair of P_e if and only if they are interlacing. Thus, if the computation does not fail, then

$$P_e = att(e)$$

At last, if the computation fails for any reason as discussed above, then e can not be planar. Thus ii is proved.

References

1. Auslander, L. and Parter, S. V., "On imbedding graphs in the plane," *J. Math. and Mech.*, vol. 10, no. 3, pp. 517-523, May, 1961.
2. Cai, J., "Fixed Point Computation and Transformational Programming," Ph.D. thesis, Rutgers University, May, 1987.
3. Hall, D. and Spencer, G., *Elementary Topology*, Wiley, New York, 1955.
4. Hopcroft, J. and Tarjan, R., "Efficient Planarity Testing," *JACM*, vol. 21, no. 4, pp. 549-568, October, 1974.
5. Mehlhorn, K., *Graph Algorithms and NP-Completeness*, Data Structures and Algorithms, 2, pp. 93-122, Springer-Verlag, 1984.
6. Paige, R. and Koenig, S., "Finite Differencing of Computable Expressions," *ACM TOPLAS*, vol. 4, no. 3, pp. 402-454, July, 1982.
7. Paige, R., "Applications of Finite Differencing to Database Integrity Control and Query/Transaction Optimization," in *Advances In Database Theory*, ed. Gallaire, H., Minker, J., and Nicolas, J.-M., vol. 2, pp. 171-210, Plenum Press, New York, Mar, 1984.
8. Thron, W.T., *Introduction to the Theory of Functions of a Complex Variable.*, Wiley, New YorK, 1953.

N NYU COMPSCI TR-324 c.2
C Cai, Jiazhen
A An interative version of
Hopcroft and Tarjan's
planarity testing